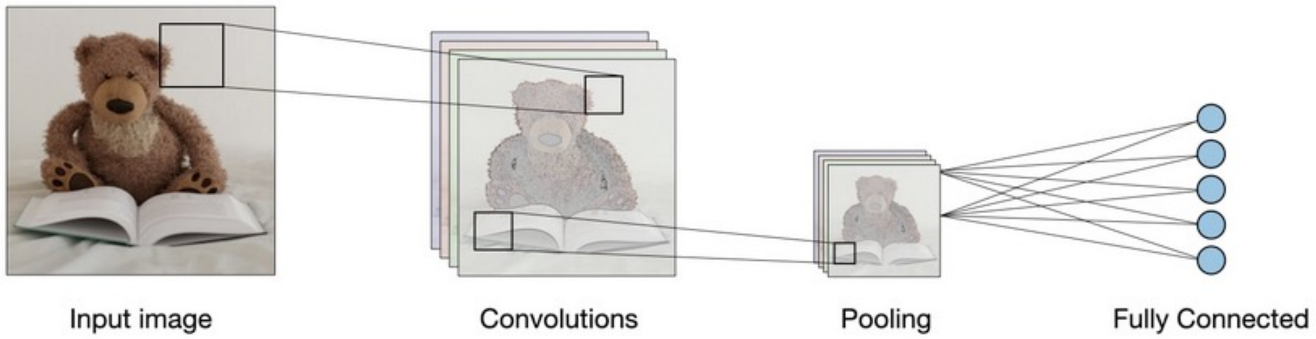
Automated Machine Learning

Frank Zijun Zhang, PhD
Division of AI in medicine, Cedars-Sinai Medical Center

UCLA QCBio Collaboratory
AutoML workshop, Winter 2023

Agenda

- Day1: Build CNNs and Model Evaluation
  - Understand how CNNs model genomic sequences
  - Build CNNs with Tensorflow and PyTorch
  - Model evaluations and tuning



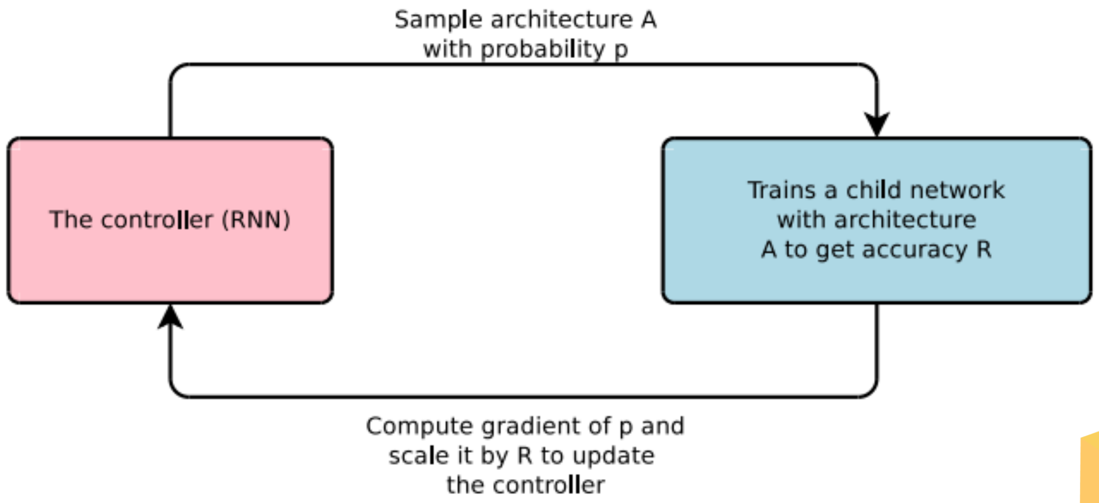
TensorFlow Keras

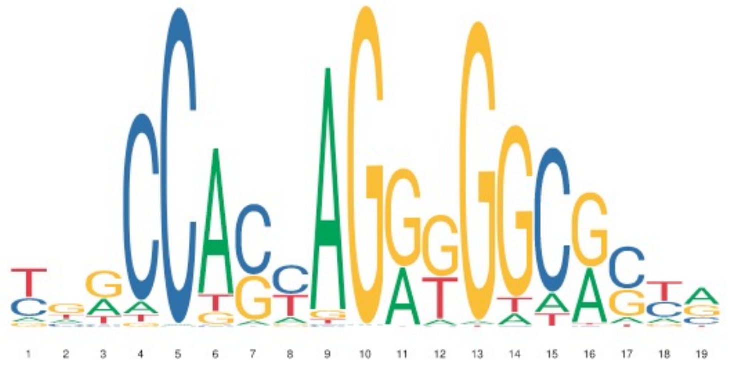
PyTorch Lightning AI

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Agenda

- Day2: AutoML and Model Explanation
  - Introduction of reinforcement learning
  - Apply NAS to automate CNN tuning
  - Interpret model and sequence motifs


Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller



AMBER

Zoph and Le, ICLR, 2017

A note on Google Colab and Github

- Go to Github Repo:
- https://github.com/zhanglab-aim/ucla-automl-workshop

- Open “UCLA_AutoML_workshop_Day2.ipynb”.

- Click on the Icon Open in Colab

- Each practice is ~10min; basic code is already implemented in the Notebook

- Explore the Practice questions shown in the slides.

Recap: Building CNNs

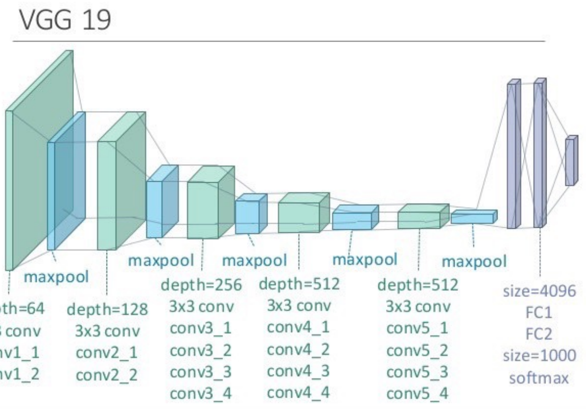
- We covered how convolution, pooling, flatten, and fully-connected layers work.
- We implemented a simple CNN model in Keras and PyTorch.
- The model achieved testing AUROC=0.79
  - Can we build a better model?

Day 2: AutoML and Model Explanation

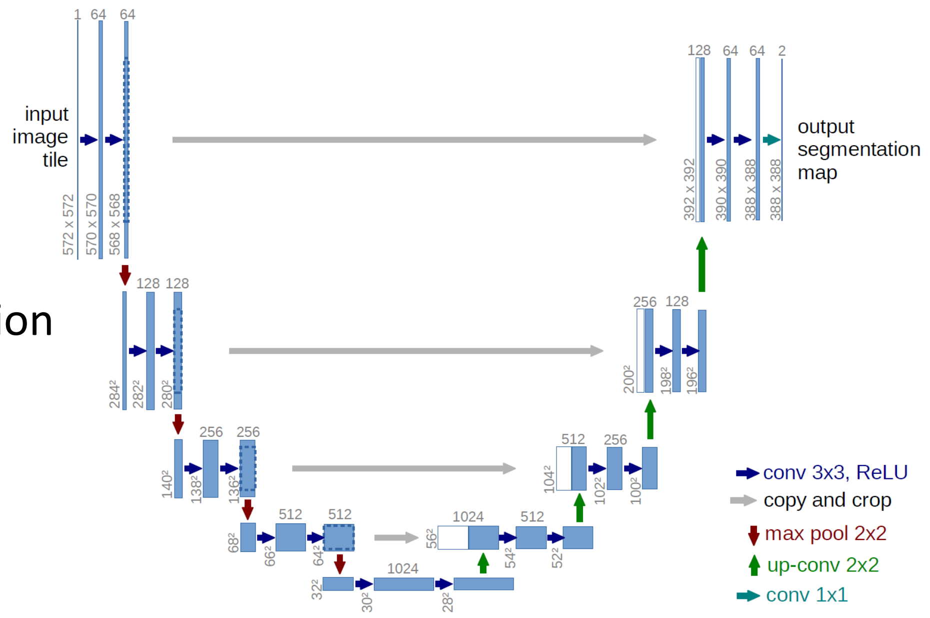
Frank Zijun Zhang, PhD
Division of AI in medicine, Cedars-Sinai Medical Center

UCLA QCBio Collaboratory
AutoML workshop, Winter 2023

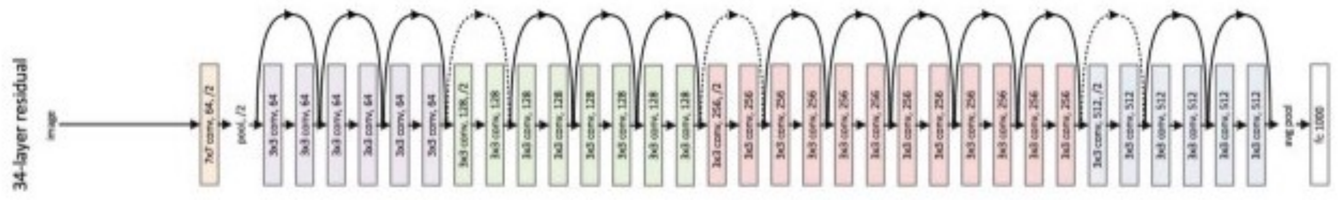
Performance of CNNs is reliant on model architectures

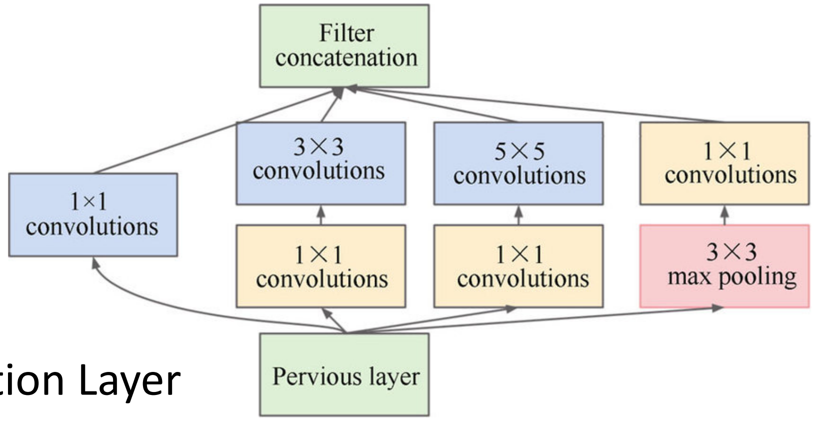


U-Net for medical image segmentation



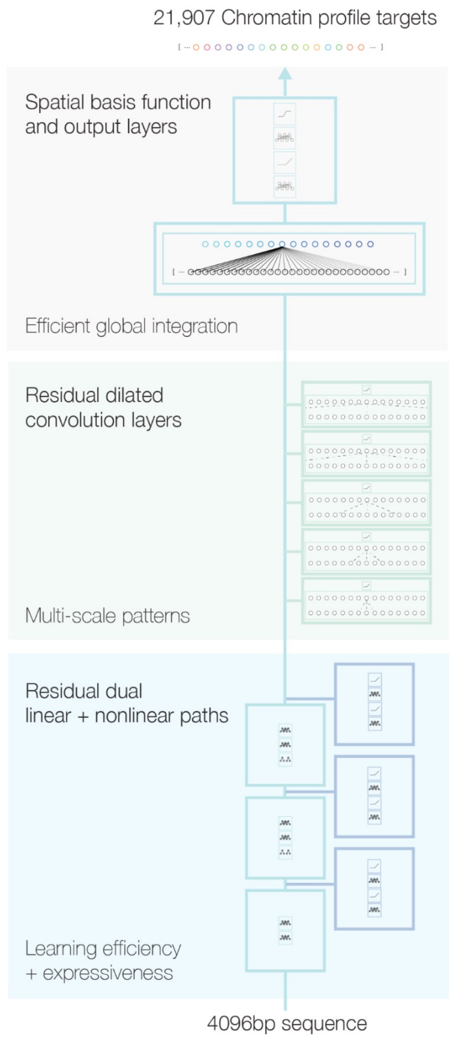
ResNet





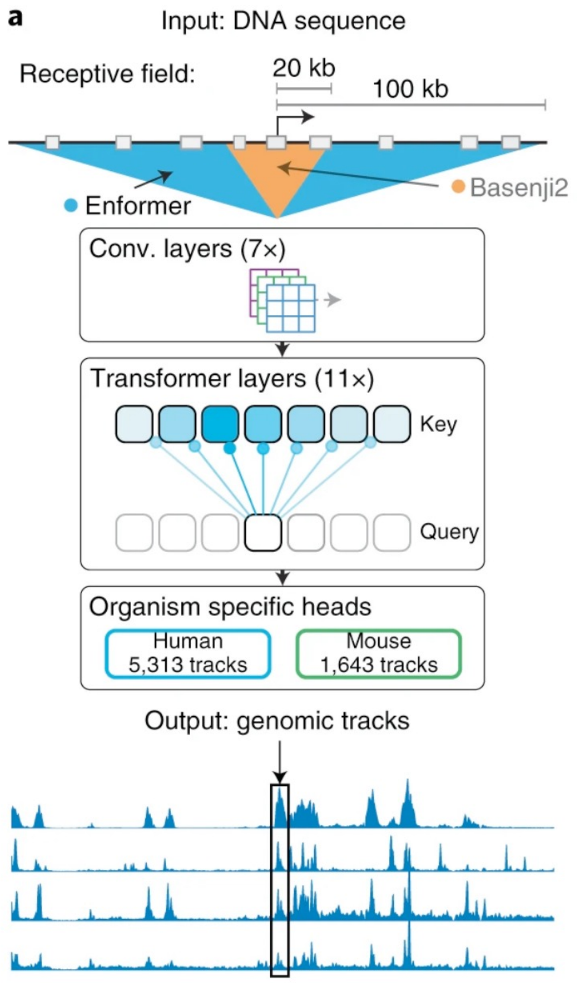
Inception Layer

Performance of CNNs is reliant on model architectures

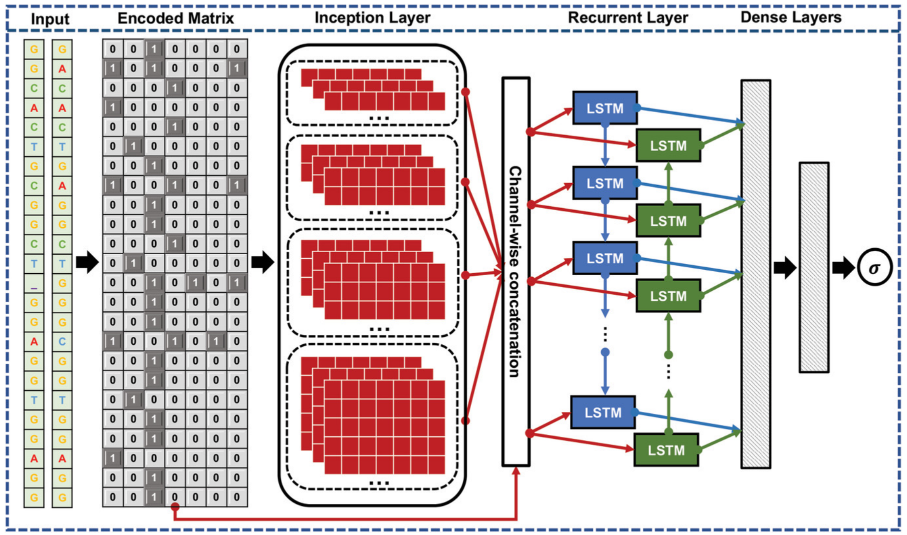


21,907 Chromatin profile targets
Spatial basis function and output layers
Efficient global integration
Residual dilated convolution layers
Multi-scale patterns
Residual dual linear + nonlinear paths
Learning efficiency + expressiveness
4096bp sequence

Sei
Chen et al., 2022



Enformer
Avsec et al., 2021



CRISPR-Net
Lin et al., 2020

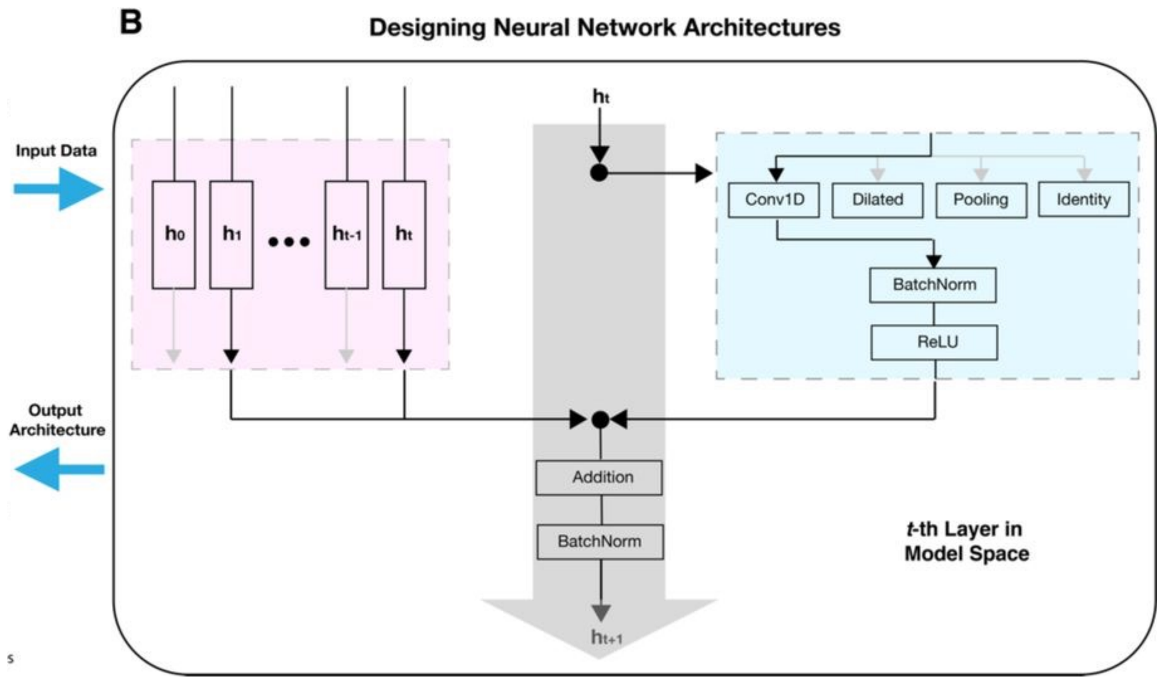
CNN architecture is important in biomedicine

- A different challenge: Biomedicine and Genomics are data-rich yet highly heterogenous.
  - e.g., single cell RNA-seq vs bulk RNA-seq may require different model complexities.
  - harder to scale-up

- High-stakes prediction: Accurate models are essential for decision-making support and knowledge discovery

What is a CNN's architecture?

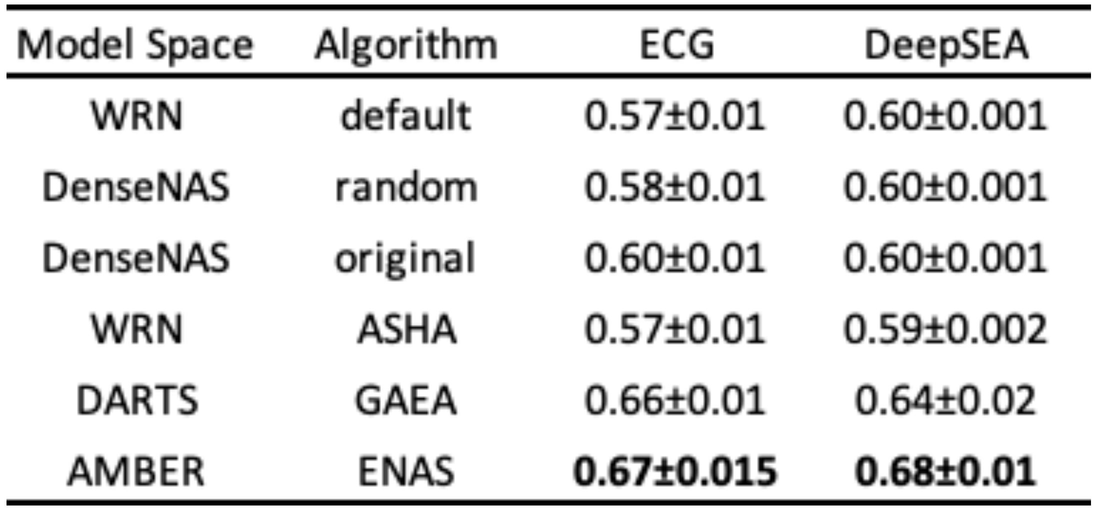
- Architecture = Operators + Wiring
  - Conv
  - Pool
  - Flatten
  - FC
- Wiring
  - Skip connections
- For simplicity, we focus on operator searching in this workshop.



AMBER
Zhang et al., 2021

Introduction to AMBER

- AMBER is a modularized AutoML framework for genomics and biomedicine, with SOTA search efficiency in 1D tasks.
- amber.architect implements AutoML methods
- amber.modeler converts a list of Python operators to Models in backend deep-learning libraries (framework-agnostic).

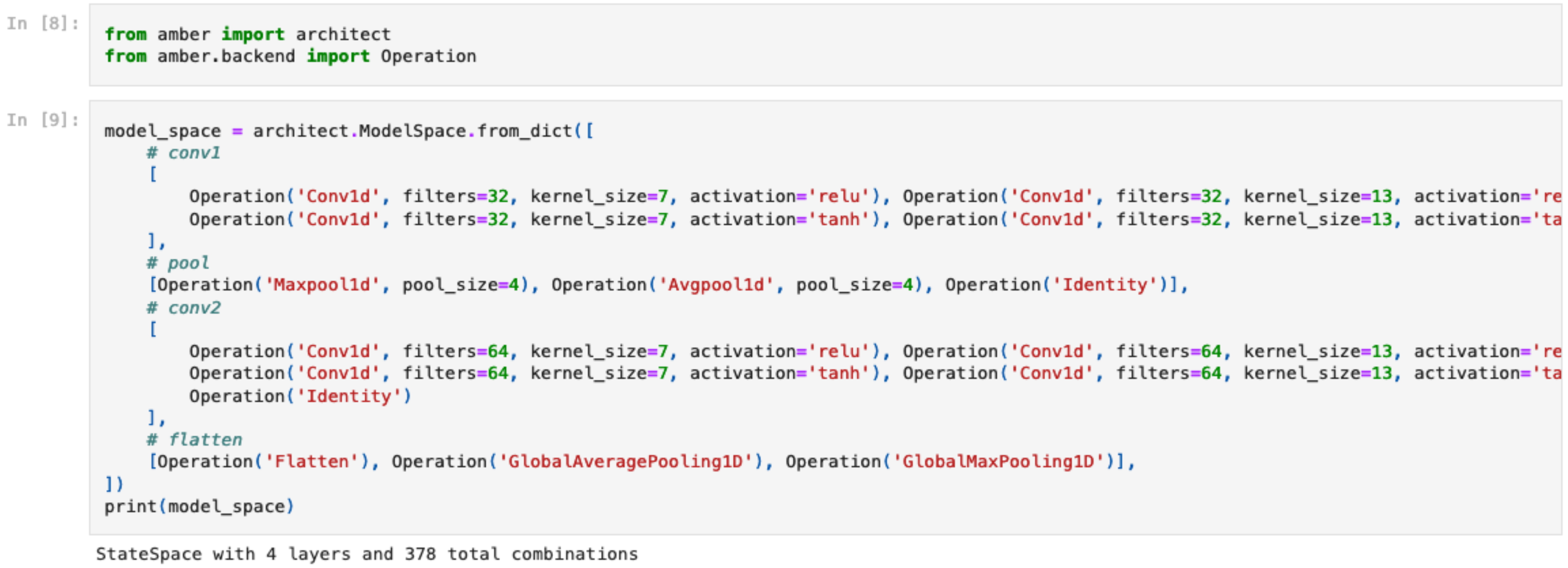


Model Space	Algorithm	ECG	DeepSEA
WRN	default	0.57±0.01	0.60±0.001
DenseNAS	random	0.58±0.01	0.60±0.001
DenseNAS	original	0.60±0.01	0.60±0.001
WRN	ASHA	0.57±0.01	0.59±0.002
DARTS	GAEA	0.66±0.01	0.64±0.02
AMBER	ENAS	**0.67±0.015**	**0.68±0.01**

NAS-Bench-360, Tu et al., NeurIPS, 2022

Model Space: a container for layer combinations

- A model space is a List of Lists
- Consider the general order: conv – pool – conv – flatten
- A special operator: *identity* will remove a layer



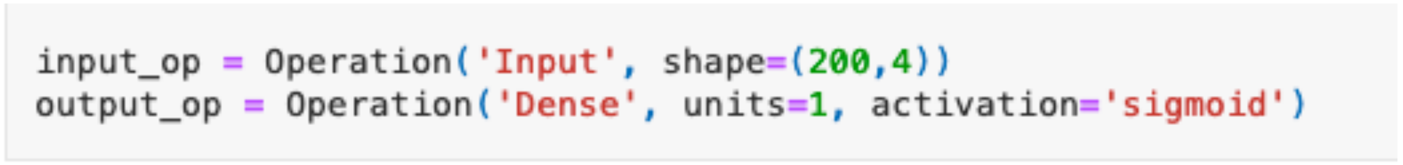
```
In [8]:  from amber import architect
         from amber.backend import Operation
```

```
In [9]:  model_space = architect.ModelSpace.from_dict([
             # conv1
             [
                 Operation('Conv1d', filters=32, kernel_size=7, activation='relu'), Operation('Conv1d', filters=32, kernel_size=13, activation='re
                 Operation('Conv1d', filters=32, kernel_size=7, activation='tanh'), Operation('Conv1d', filters=32, kernel_size=13, activation='ta
             ],
             # pool
             [Operation('Maxpool1d', pool_size=4), Operation('Avgpool1d', pool_size=4), Operation('Identity')],
             # conv2
             [
                 Operation('Conv1d', filters=64, kernel_size=7, activation='relu'), Operation('Conv1d', filters=64, kernel_size=13, activation='re
                 Operation('Conv1d', filters=64, kernel_size=7, activation='tanh'), Operation('Conv1d', filters=64, kernel_size=13, activation='ta
                 Operation('Identity')
             ],
             # flatten
             [Operation('Flatten'), Operation('GlobalAveragePooling1D'), Operation('GlobalMaxPooling1D')],
         ])
         print(model_space)

         StateSpace with 4 layers and 378 total combinations
```

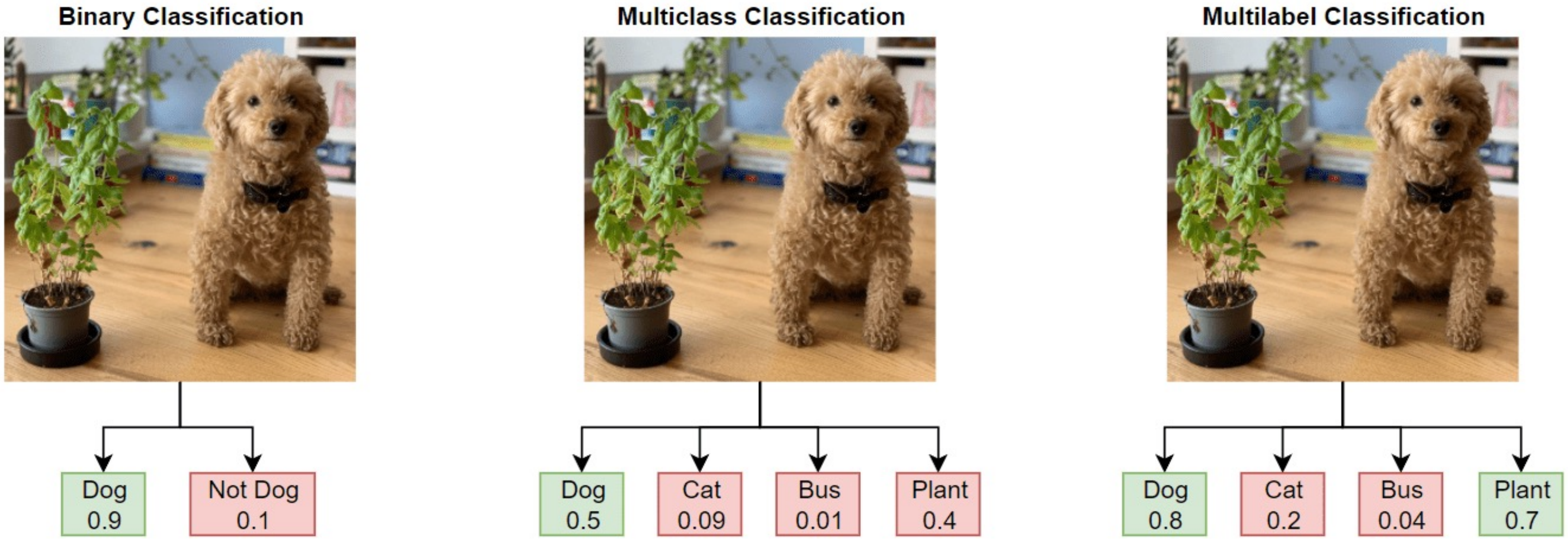
Model Space: a container for layer combinations

- Aside from the moving parts, we need fixed input and output operators.
- A rule of thumb: observe the input feature and output label shapes.
  - For output, find a proper activation function, too.
  - continuous regression: activation=‘linear’
  - binary/multi-label classification: activation=‘sigmoid’
  - multi-class classification: activation=‘softmax’



```
input_op = Operation('Input', shape=(200,4))
output_op = Operation('Dense', units=1, activation='sigmoid')
```

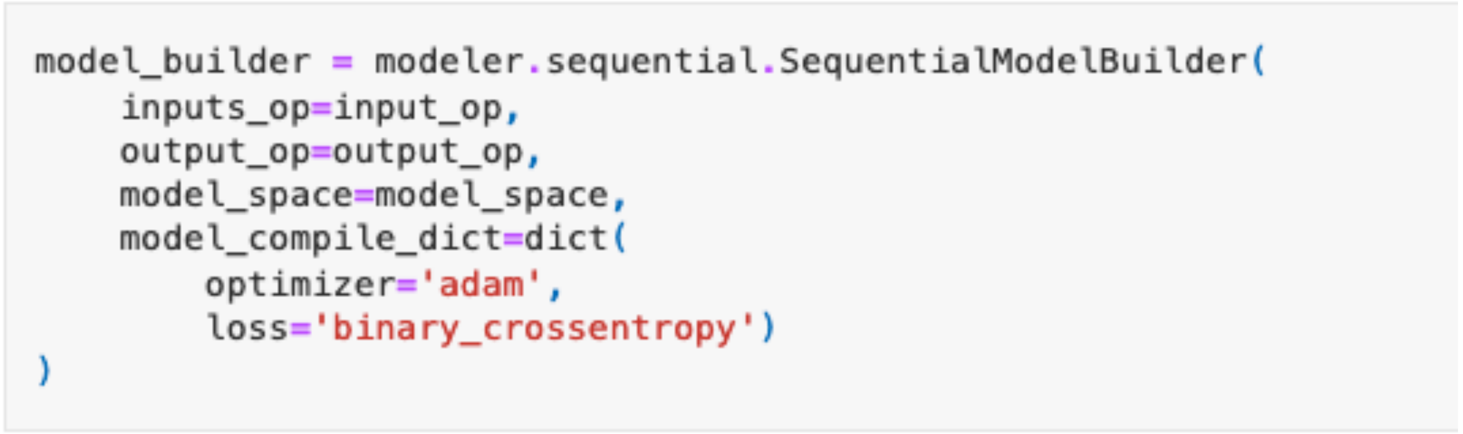
Multi-label vs Multi-class



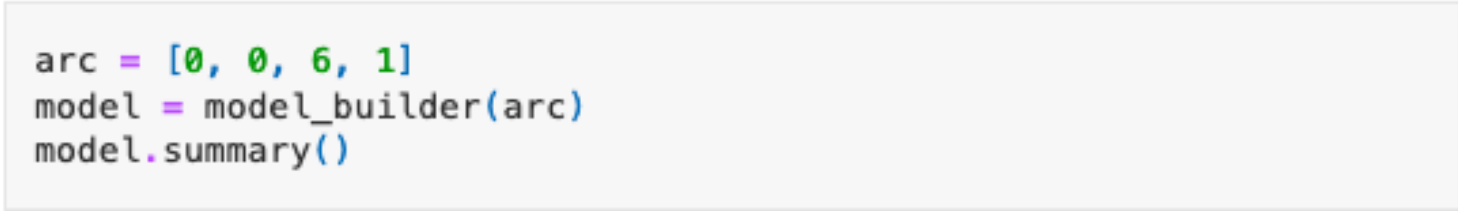
https://www.mathworks.com/help/deeplearning/ug/multilabel-image-classification-using-deep-learning.html

Sampling a CNN from Model Space

- Now we have input/output ops and a viable Model Space, we can sample a list of architecture tokens that translates to a CNN.
- This is implemented as modeler.sequential.SequentialModelBuilder
  - input arc= [0, 0, 6, 1]
  - output a CNN model (the model we built in Day 1)



```
model_builder = modeler.sequential.SequentialModelBuilder(
    inputs_op=input_op,
    output_op=output_op,
    model_space=model_space,
    model_compile_dict=dict(
        optimizer='adam',
        loss='binary_crossentropy')
)
```



```
arc = [0, 0, 6, 1]
model = model_builder(arc)
model.summary()
```

Sampling a CNN from Model Space

- More importantly, SequentialModelBuilder is framework-agnostic.
- You can switch the backend by !amber-cli config -b {pytorch, tensorflow_2} and build CNNs under different frameworks, without changing any code.

- AMBER enables a unified syntax between Tensorflow 1/2 and PyTorch.

Practice 1. Model Space and CNNs

Run Sections 1-2. Define Model Space & Build CNN Open in Colab

- Change the arc tokens and/or model_compile_dict. Does this increase or decrease your model's performance?

- Change the backend from PyTorch to Tensorflow 2 and re-run Sections 1-2.

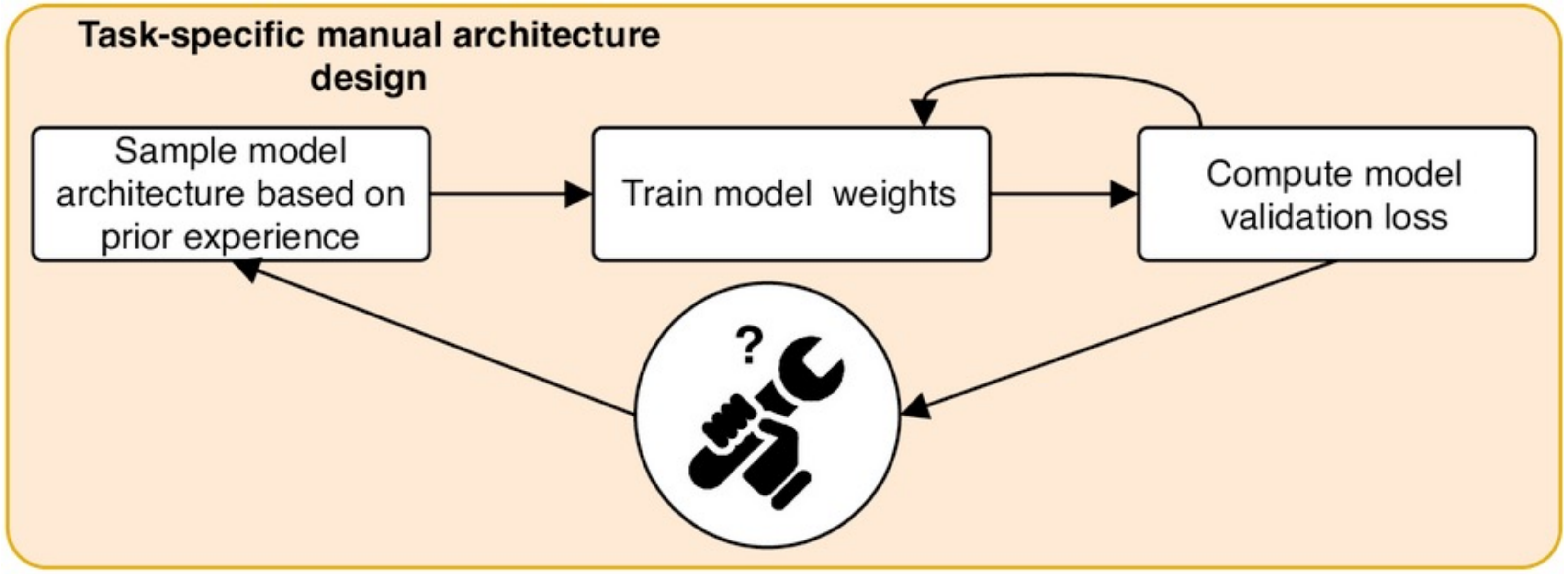
- Add another layer of Dense operators in architect.ModelSpace after the Flatten layer.

  hint: Operation('Dense', units=16, activation='relu')

  if you change the model space, do you need to change the arc tokens when calling model_builder?

Manual Tuning of Model Architectures

- As you may have notices, changing the arc tokens will build different CNNs with varying predictive powers.


Task-specific manual architecture design
Sample model architecture based on prior experience
Train model weights
Compute model validation loss
?

Zhang, Cofer and Troyanskaya, 2021

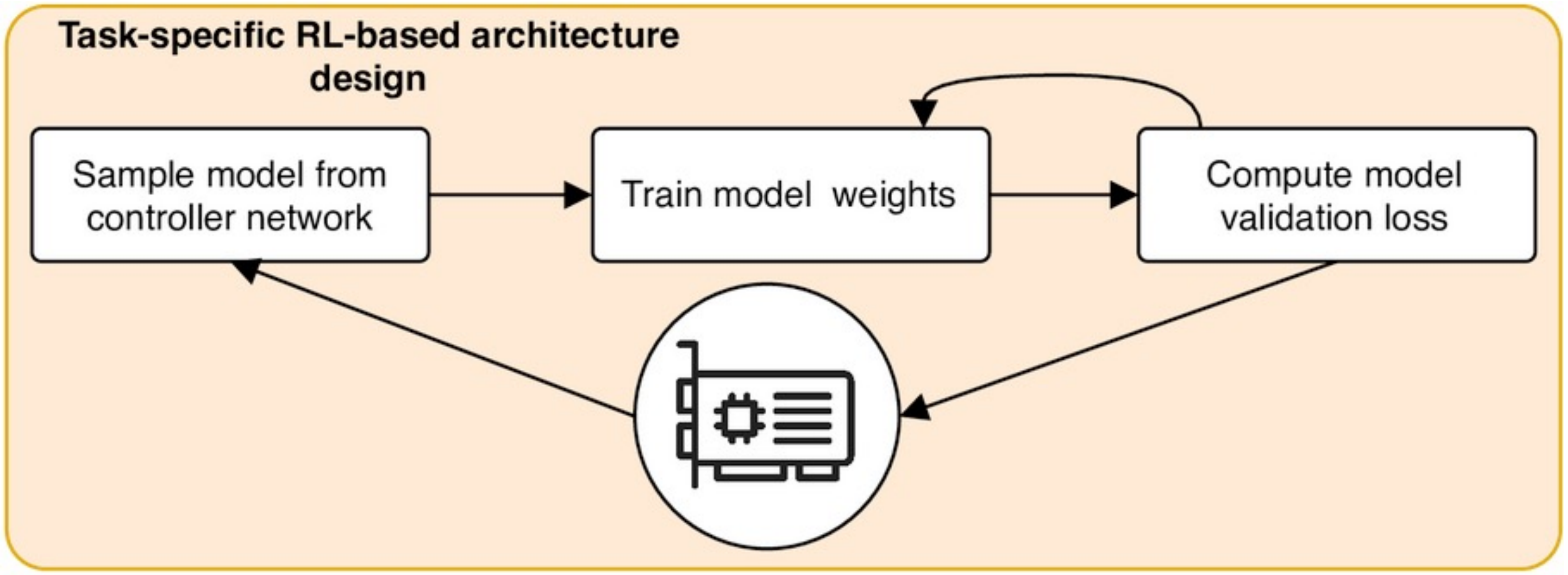
Automated model building process

- At the bare minimum, we can use a for-loop to randomly sample CNNs within a given time budget, and choose the best model.

- However, we can do better by building another model that learns from the history of CNN models to propose arc tokens better than random.

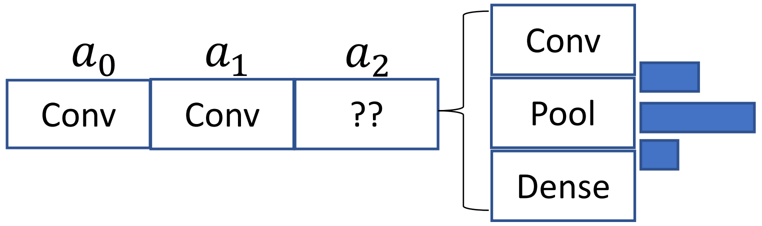
Reinforcement-learning Tuning of Model Architectures

- Reinforcement-learning (RL): reward desired CNN architectures, and penalize underperforming architectures.


Task-specific RL-based architecture design
Sample model from controller network
Train model weights
Compute model validation loss

Neural Architecture Search with Reinforcement Learning

- A model-based approach: a recurrent neural network (RNN) **controller** model that learns the arc tokens as a sequence.

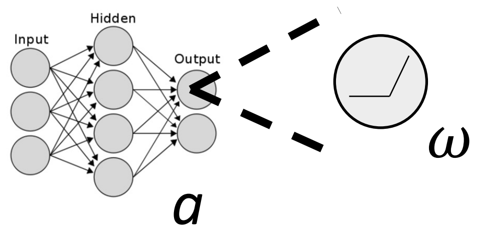


https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

Math formulations of controller basics

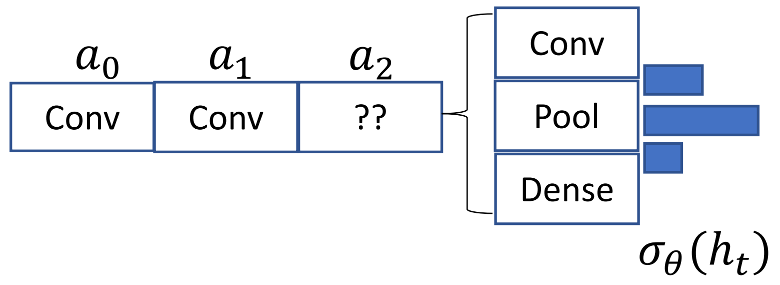
1. To learn a function that maps *x* to *y*, optimize its architectures *a*:

$$y_i = f_{\omega;a}(x_i)$$



2. Sample a_t from the conditional probability $P(a_t|a_{t-1}, \dots, a_0)$ by a Recurrent Neural Network $\sigma_\theta(\cdot)$ with parameters θ:

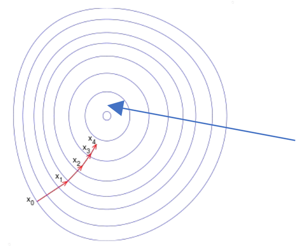
$$a_t \sim P(a_t|a_{t-1}, \dots, a_0) = \sigma_\theta(h_t)$$



$\pi(a_k;\theta)$: log-likelihood of a_k
R_k: reward for a_k
b: moving average of *R*

3. Optimize θ w.r.t. to a reward *R* (usually validation accuracy):

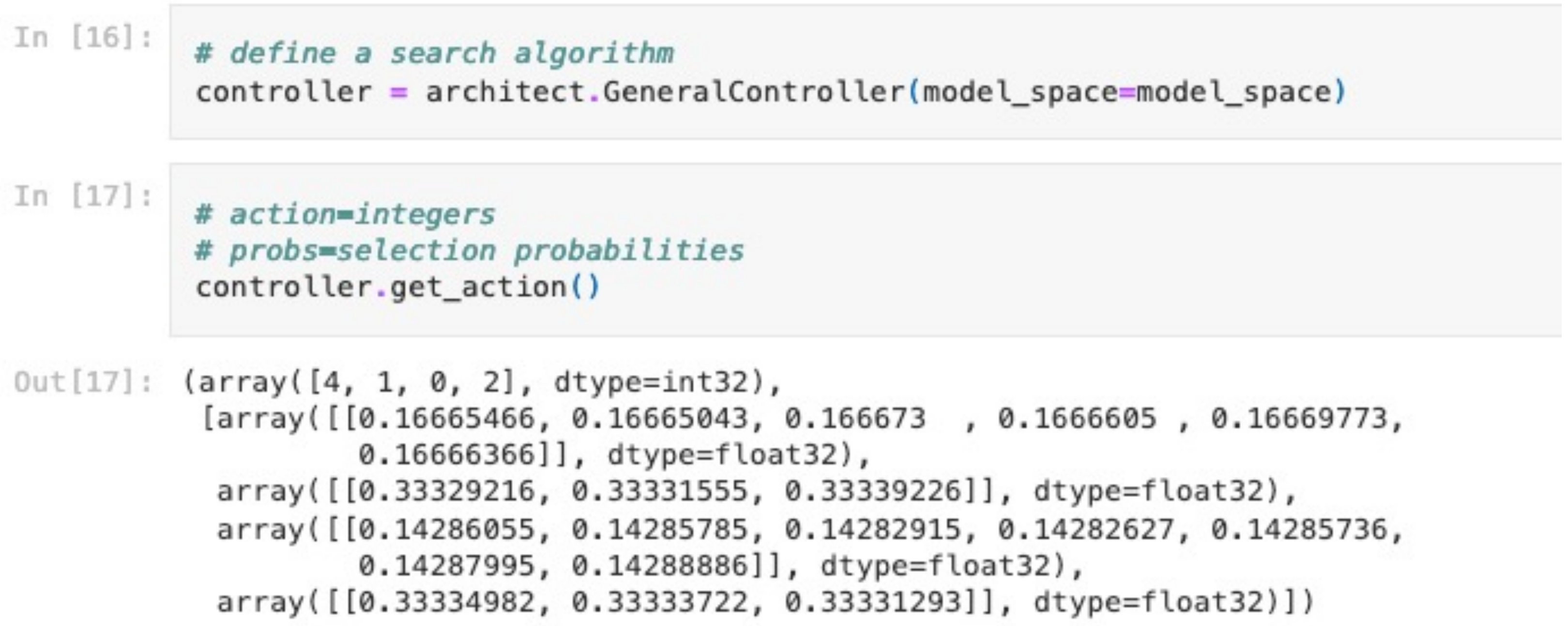
$$\frac{1}{m}\sum_{k=1}^{m} \nabla_\theta \pi(a_k;\theta)(R_k - b)$$



Objective: high reward with large likelihood

Controller implementation in AMBER

- A controller can be built by parsing a Model Space.
- Initially, the controller is naive and will sample architecture tokens uniformly.
- As we will see, over time these probs are skewed towards better rewards.

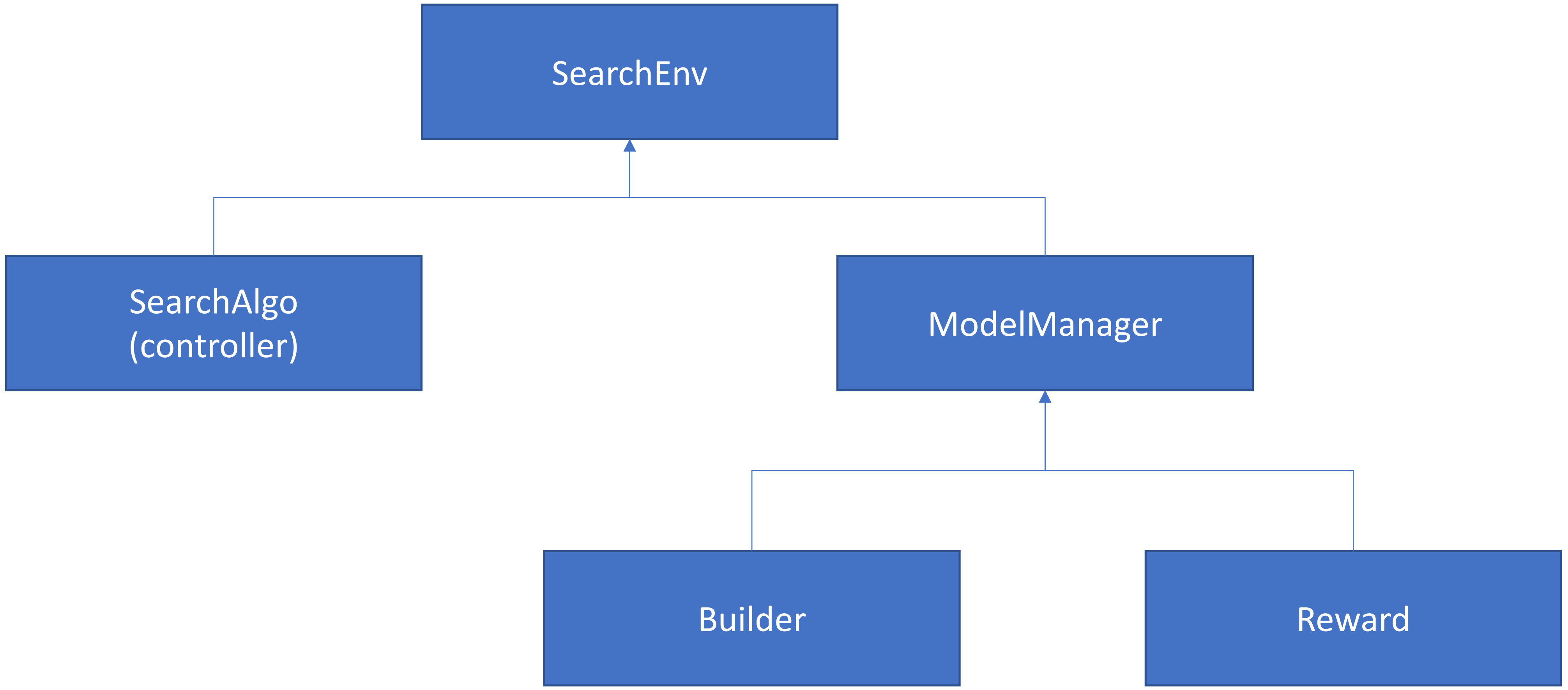


```
In [16]:  # define a search algorithm
          controller = architect.GeneralController(model_space=model_space)
```

```
In [17]:  # action=integers
          # probs=selection probabilities
          controller.get_action()
```

```
Out[17]: (array([4, 1, 0, 2], dtype=int32),
           [array([[0.16665466, 0.16665043, 0.166673  , 0.1666605 , 0.16669773,
                    0.16666366]], dtype=float32),
            array([[0.33329216, 0.33331555, 0.33339226]], dtype=float32),
            array([[0.14286055, 0.14285785, 0.14282915, 0.14282627, 0.14285736,
                    0.14287995, 0.14288886]], dtype=float32),
            array([[0.33334982, 0.33333722, 0.33331293]], dtype=float32)])
```

Components of AMBER NAS



Practice 2. Running AMBER NAS

Run Section 3. Automated Search Open in Colab
Section 4. Understanding AutoML

- Remember to change your Colab Runtime type to “GPU”.
  - should finish in ~1min.

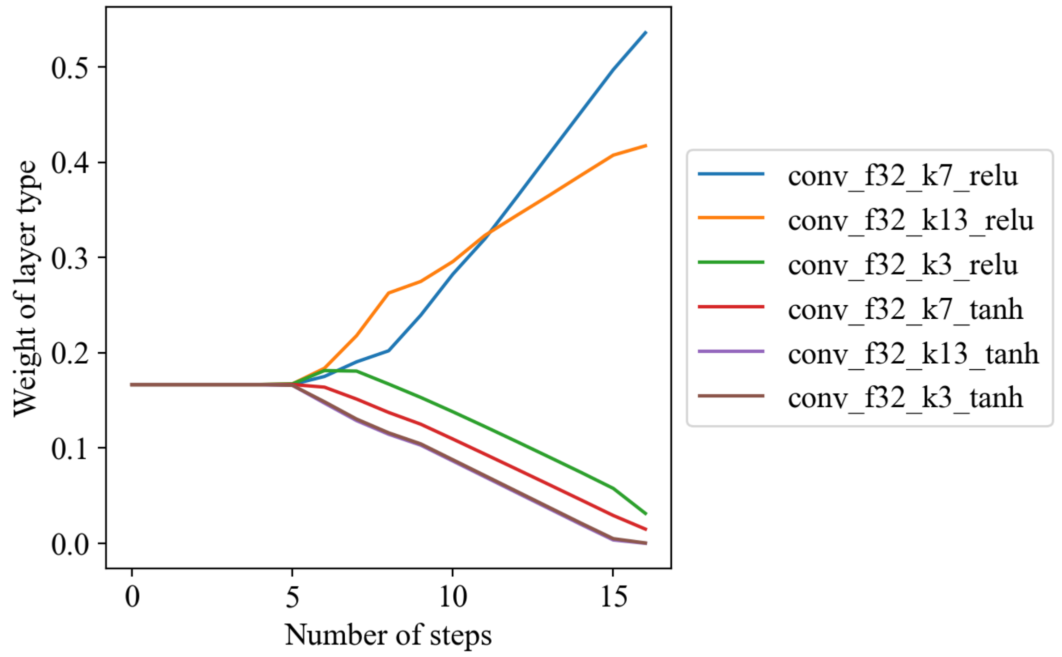
- Change the backend from PyTorch to Tensorflow_2.

- Change the method of reward_fn from “auroc” to a custom callable function to compute Accuracy acc; restart the runtime and run all. How does the performance compare?
  - `acc = lambda y_true, y_score: np.mean(y_true==(y_score>0.5).astype(int))`

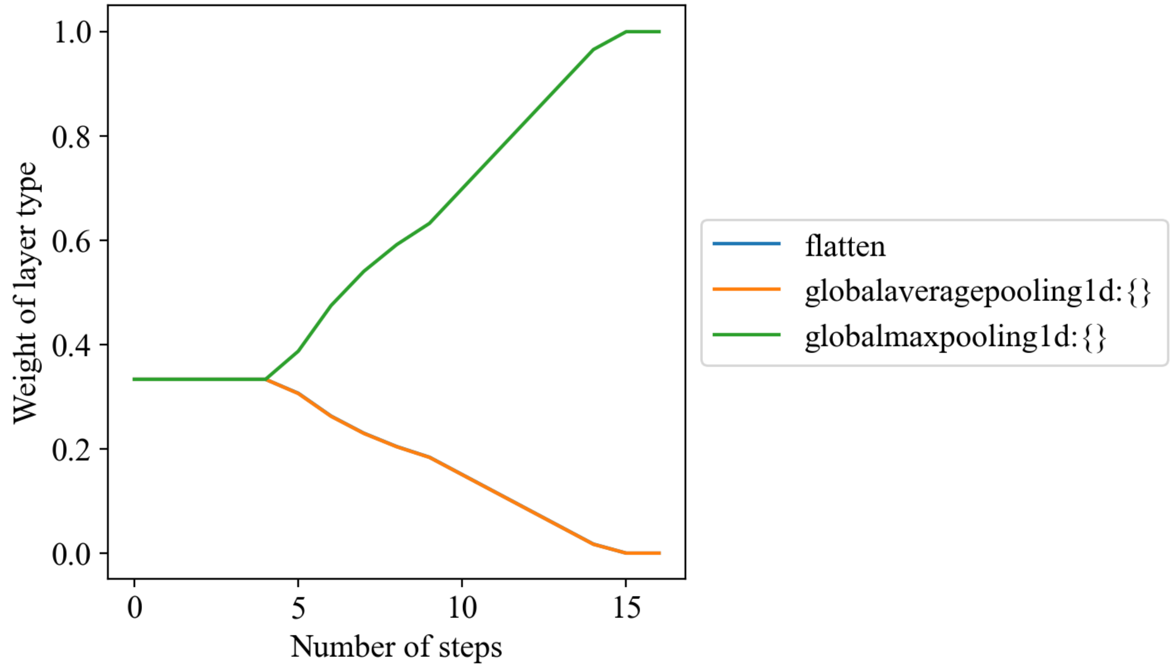
Understanding NAS Principles

- Data-driven model building can reveal insights for our dataset, and for future modeling.
- Sometimes, it can even tell us when a model is too complex for the dataset by selecting *identity* over other operators.

Conv Layer

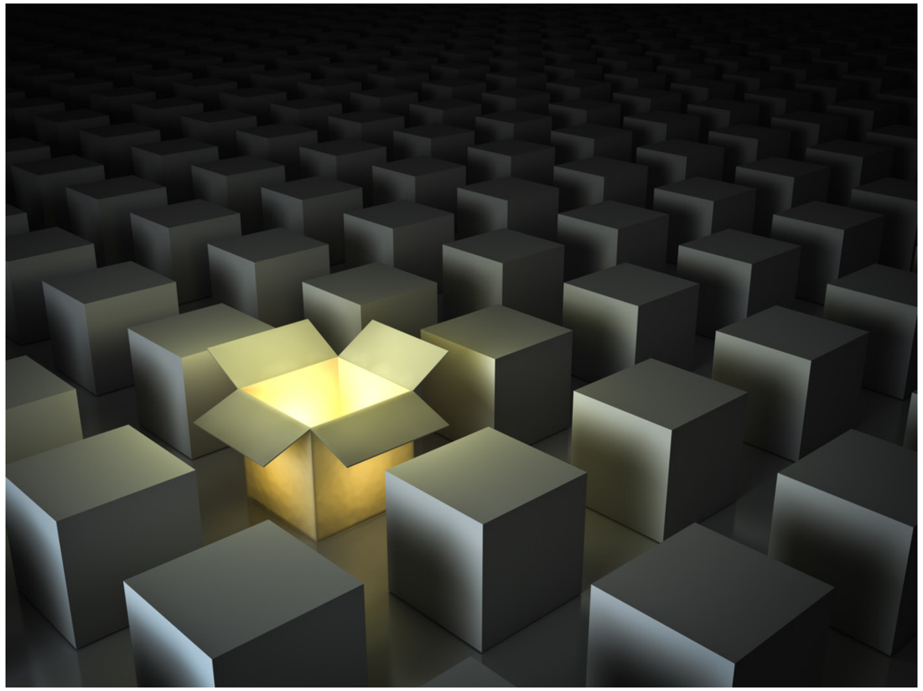


Flatten Layer



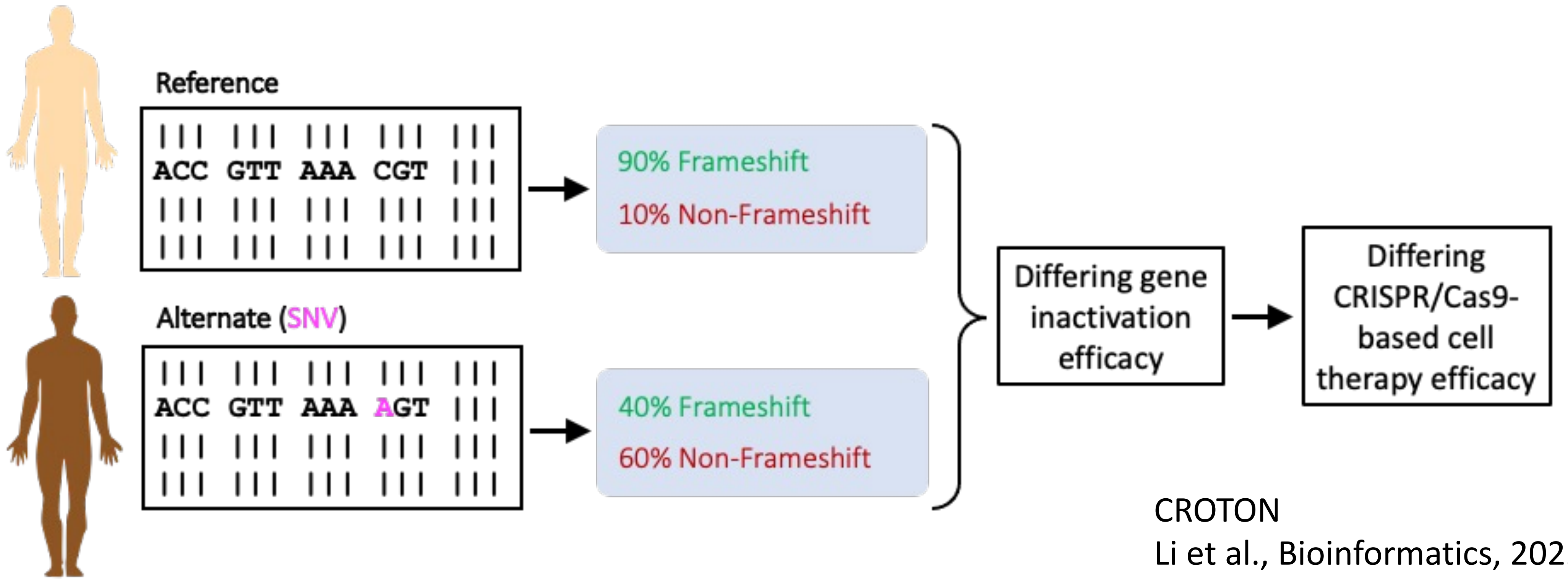
Interpreting the trained model

- We can use interpretation methods to understand what evidence is useful for our model.
- For genomic sequences, a popular approach is in-silico mutagenesis (ISM).


(amasterphotographer/Shutterstock)

Interpreting the trained model

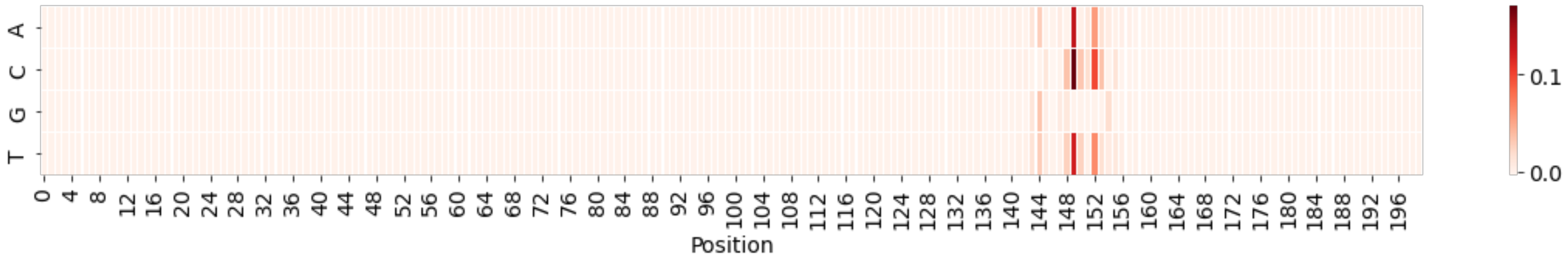
- in-silico mutagenesis (ISM) introduces mutations to the input sequence, even though such mutations can be rare in natural populations and thus lack of power for analysis.



CROTON
Li et al., Bioinformatics, 2022

Saturated ISM

- Saturated ISM exhaustively perturbs every base-pair.
- The resulting score matrix is the same shape as the input sequence.



Practice 3. Model Explanation

Run Section 5.ISM for Model Explanation Open in Colab

- Pick the second-best model and analyze its model explanation results. Is it the same with the best model?

- Compare the focus region with the ground truth motif.

- Think of ways to compute the prediction difference when two letters are changed.

Summary

- Introduction of reinforcement learning
- Apply NAS to automate CNN tuning
- Interpret model and sequence motifs

- AutoML and its application in Genomics and Biomedicine is an exciting research field.
  - Feel free to reach out to me if you'd like to explore more in this direction.

Q&A

Write your questions in this [Google Doc].